DISEÑO DE PRUEBAS UNITARIAS

Danna A. Espinosa

Carlos J. Bolaños

Cristian F. Perafan

Computación y Estructuras Discretas I

Universidad Icesi

Profesor: Uram Anibal Sosa

TEST CASE DESIGN

Configuración de los Escenarios

Nombre	Clase	Escenario
setUpStage1	GraphTest	En primera instancia se crea el grafo, posteriormente se agregan 1 vértice de tipo “Integer” al grafo.
setUpStage2	GraphTest	En primera instancia se crea el grafo, posteriormente se agregan mil vértices de tipo “Integer” al grafo.
setUpStage3	GraphTest	En primera instancia se crea el grafo, posteriormente se agregan 5 vértices de tipo “Integer” al grafo y se agrega 7 aristas al grafo.
setUpStage4	GraphTest	Se agregan 4 vértices con sus respectivas aristas y pesos.
setUpStage5	GraphTest	Se instancian ciudades para comprobar el funcionamiento del algoritmos de Dijkstra y Floyd - Warshall

Diseño de Casos de Prueba

Prueba #1:

Objetivo de la Prueba: verificar que el método buscar del grafo funcione correctamente.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Graph	searchANode()	setUpStage1()	Integer v = 1	El resultado de búsqueda una vez se ejecute el método de buscar un nodo debe ser igual al valor de entrada.

Prueba #2:

Objetivo de la Prueba: verificar que se puedan insertar vértices del grafo, esta verificación se realiza con la ayuda del método buscar un vértice.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Graph	insertANode()	setUpStage1()	Integer v = 1	El resultado de búsqueda una vez el nodo se halla agregado debe ser diferente de null.

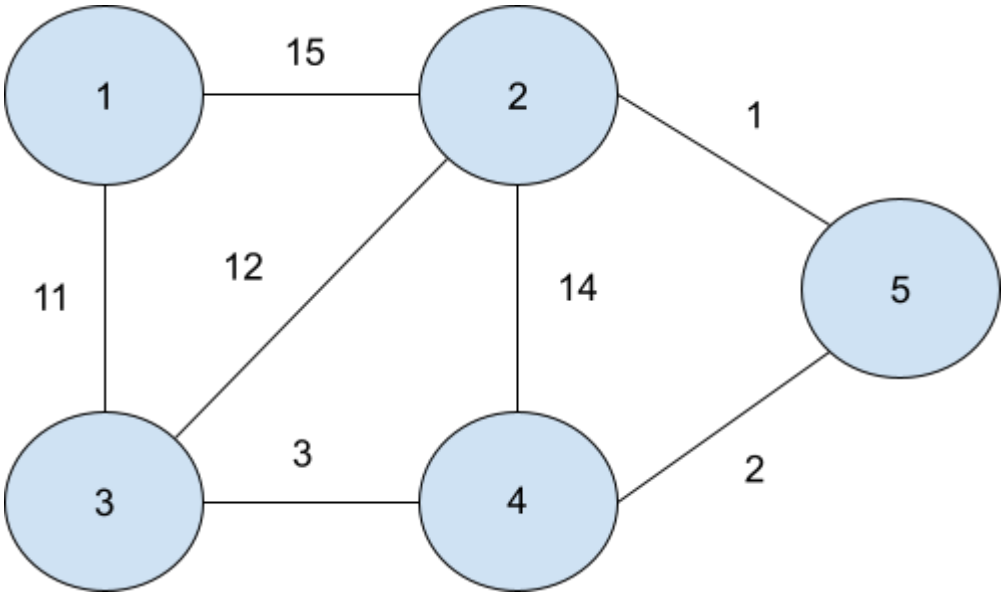
Prueba #3:

Objetivo de la Prueba: verificar que se puedan eliminar vértices del grafo, esta verificación se realiza con la ayuda del método buscar un vértice.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Graph	deleteNode()	setUpStage2()	Integer v = 5	El resultado de búsqueda una vez el nodo se haya eliminado debe ser null

Prueba #4:

Objetivo de la Prueba: verificar que se pueden agregar aristas al grafo, para esta implementación se utlizan el escenario setUpStage2().				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Graph	testInsertAEdge()	setUpStage2()	Weight = 20 Integer vertex_1 = 5 Integer vertex_2 = 500	El resultado del método test InsertA Edge() debe ser verdadero

TEST CASES DESIGN FLOYD - WARSHALL ALGORITHM



Prueba #5:

Objetivo de la Prueba: verificar el correcto funcionamiento del algoritmo Floyd-Warshall, para esta prueba se utiliza el escenario setUpStage2().				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Graph	testFloydWarshallAlgorithm()	setUpStage2()	N/A	El algoritmo debe retornar la siguiente matriz con la distancia mínima entre cada par de grafos.

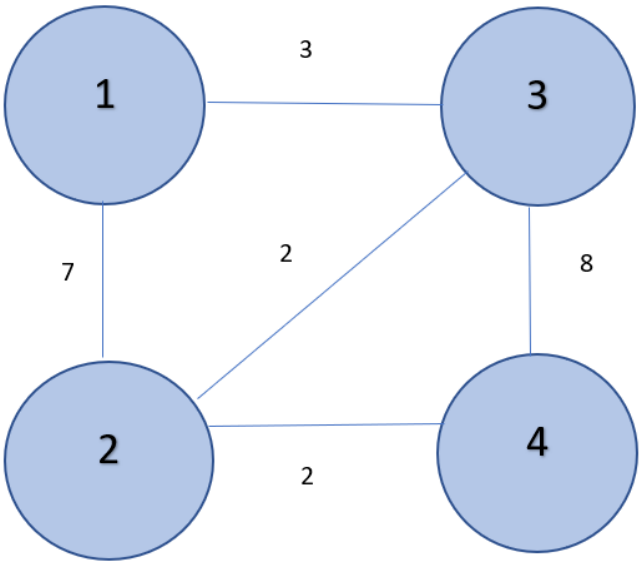
Matriz:

		j				
		1	2	3	4	5
i	1	0	15	11	14	16
	2	15	0	6	3	1
	3	11	6	0	3	5
	4	14	3	3	0	2
	5	16	1	5	2	0

Prueba #6:

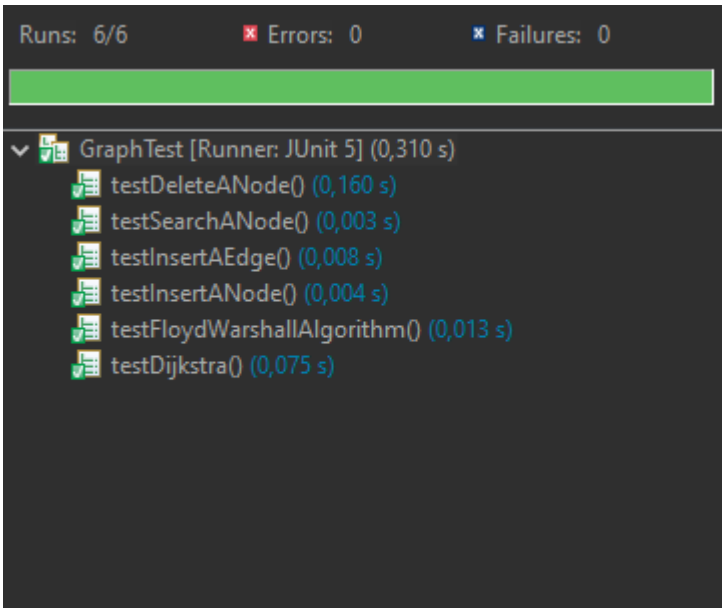
Objetivo de la Prueba: verificar el correcto funcionamiento del algoritmo de Dijkstra implementado.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Graph	testDijkstra()	setUpStage4()	Node<Integer> node1= 1 Node<Integer> node2 = 2	El resultado del método testDijkstra debe ser igual a la Stack quemada en el código de test.

TEST CASES DESIGN DIJKSTRA ALGORITHM



Objetivo de la Prueba: verificar el correcto funcionamiento del algoritmo de Dijkstra implementado.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Graph	testDijkstra()	setUpStage4()	Node<Integer> node1= 1 Node<Integer> node2 = 2	El resultado del método testDijkstra() debe ser igual a la Stack quemada en el código de test.

Resultados JUnit5:



prueba #7:

Objetivo de la Prueba: Verificar el correcto funcionamiento del Algoritmo de Dijkstra con las entradas originales del programa.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Graph	testTravelContries()	setUpStage5()	Node<String> node1= Buenos Aires Node<String> node2 = Bogotá	El resultado del método testTravelContries() debe ser igual a la Stack quemada en el código de test.